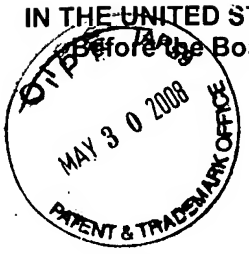
IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
Before the Board of Patent Appeals and Interferences

1FW AF



In re Patent Application of

FRANCIS, H. et al.

Serial No. 10/648,293

Filed: August 27, 2003

Atty Dkt. JRL-550-450
C# M#
Confirmation No. 4462

TC/A.U.: 2183

Examiner: Pan, Daniel H.

Date: May 30, 2008

Title: EXECUTING VARIABLE LENGTH INSTRUCTIONS STORED WITHIN A PLURALITY OF DISCRETE MEMORY ADDRESS REGIONS

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

☐ **Correspondence Address Indication Form Attached.**

☐ **NOTICE OF APPEAL**
Applicant hereby **appeals** to the Board of Patent Appeals and Interferences from the last decision of the Examiner twice/finally rejecting applicant's claim(s). $510.00 (1401)/$255.00 (2401) $

☒ An appeal **BRIEF** is attached in the pending appeal of the above-identified application **Fee Previously Paid January 18, 2008.** $510.00 (1402)/$255.00 (2402) $

☐ Credit for fees paid in prior appeal without decision on merits -$ ( )

☒ **A response to Notice of Non-Compliant Appeal Brief is attached.** **(no fee)**

☐ Petition is hereby made to extend the current due date so as to cover the filing date of this paper and attachment(s)
One Month Extension $120.00 (1251)/$60.00 (2251)
Two Month Extensions $460.00 (1252)/$230.00 (2252)
Three Month Extensions $1050.00 (1253/$525.00 (2253)
Four Month Extensions $1640.00 (1254/$820.00 (2254) $

☐ "Small entity" statement attached.

Less month extension previously paid on -$( )

TOTAL FEE ENCLOSED $ 0.00

☐ **CREDIT CARD PAYMENT FORM ATTACHED.**

Any future submission requiring an extension of time is hereby stated to include a petition for such time extension. The Commissioner is hereby authorized to charge any deficiency, or credit any overpayment, in the fee(s) filed, or asserted to be filed, or which should have been filed herewith (or with any paper hereafter filed in this application by this firm) to our **Account No. 14-1140.** A duplicate copy of this sheet is attached.

901 North Glebe Road, 11th Floor
Arlington, Virginia 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100
JRL:maa

NIXON & VANDERHYE P.C.
By Atty: John R. Lastova, Reg. No. 33,149

Signature: _____

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

FRANCIS, H. et al. Atty. Ref.: 550-450; Confirmation No. 4462

Appl. No. 10/648,293 TC/A.U. 2183

Filed: August 27, 2003 Examiner: Pan, Daniel H.

For: EXECUTING VARIABLE LENGTH INSTRUCTIONS STORED WITHIN A PLURALITY OF DISCRETE MEMORY ADDRESS REGIONS

* * * * * * * * * * * *

May 30, 2008

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

RESPONSE TO NOTIFICATION OF NON-COMPLIANT APPEAL BRIEF

In response to the Notification of Non-Compliant Appeal Brief (dated May 2, 2008), Applicants submit the following response along with a revised appeal brief.

The Notice of Non-Compliant Appeal Brief alleges that the Appealed Claims have not been identified in the Status of the Claims section of the brief and each ground of the rejection has not been identified. The Status of Claims section has been revised to identify appealed claims 1-62. The Argument section has been revised to identify the grounds listed in the Grounds of Rejection section This should overcome the objection noted in the Notice of Non-Compliance.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____
John R. Lastova
Reg. No. 33,149

JRL:maa
901 North Glebe Road, 11th Floor
Arlington, VA 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of

FRANCIS, H. et al.

Atty. Ref.: 550-450

Serial No. 10/648,293

Group: 2183

Filed: August 27, 2003

Examiner: Pan, Daniel H.

For: EXECUTING VARIABLE LENGTH INSTRUCTIONS STORED WITHIN A PLURALITY OF DISCRETE MEMORY ADDRESS REGIONS

Before the Board of Patent Appeals and Interferences

BRIEF FOR APPELLANT

On Appeal From Final Rejection
From Group Art Unit 2183

John R. Lastova
NIXON & VANDERHYE P.C.
11th Floor, 901 North Glebe Road
Arlington, Virginia 22203-1808
(703) 816-4025
Attorney for Appellants
Francis et al. and
ARM Limited

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Patent Application of

FRANCIS, H. et al. Atty. Ref.: 550-450

Serial No. 10/648,293 Group: 2183

Filed: August 27, 2003 Examiner: Pan, Daniel H.

For: EXECUTING VARIABLE LENGTH INSTRUCTIONS STORED WITHIN A PLURALITY OF DISCRETE MEMORY ADDRESS REGIONS

* *

May 30, 2008

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

I. REAL PARTY IN INTEREST

The real party in interest is the assignee, ARM Limited, a United Kingdom corporation.

II. RELATED APPEALS AND INTERFERENCES

There are no other appeals related to this subject application. There are no interferences related to this subject application.

III. STATUS OF CLAIMS

Claims 1-62 are pending. Claims 43-62 stand rejected under 35 U.S.C. §101. Claims 1-62 stand rejected under 35 U.S.C. §103. All rejections and all claims 1-62 are appealed.

IV. STATUS OF AMENDMENTS

No amendment has been filed after final. A pre-appeal was filed on October 29, 2007.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

An example of data processors that execute variable length instructions is the Jazelle-enabled processors produced by ARM Limited. These processors can execute programs formed of Java bytecodes that vary in their byte length. In some circumstances, it is desirable to allow a program formed of variable length instructions to be stored in several different memory regions, which means that the program is "fragmented" in memory. In a secure system, such fragmentation may be desirable as a way of obfuscating the computer program concerned. Fragmentation may also be desirable in order to better use the memory available by filling all portions of that memory even if they are discrete.

Figure 1 illustrates a problem which can arise in such fragmentation systems. A variable length program instruction is shown which is three bytes in

length and comprises a bytecode BCX followed by two operands OpdXOpdX'. This 3-byte instruction spans two discrete memory regions with the first byte being in a current memory region and the second and third bytes being in a following memory region. Standard hardware for executing such variable length instructions assumes all the bytes of a variable length instruction will be found at sequentially adjacent memory addresses. In the case illustrated in Figure 1, the standard hardware would assume that the second and third bytes of the variable length instruction immediately followed the first byte of the variable length instruction. But in fact, they are in a discrete memory region separated from the first byte.

To solve these problems, the instruction data from portions of the discrete memory regions containing the variable length instruction which spans the boundary are concatenated, and this concatenated data are stored within a fix-up memory region. The variable length instruction can then be executed from its location within this fix-up region, after which, program flow can be returned to the normal memory region following the gap between the discrete memory regions. Figure 4 provides an example illustration. This mechanism effectively deals with fragmented program code comprising variable length instructions for which the start address of the instruction following that spanning the gap will not be known until the instruction spanning the gap has been executed. At the same time, the software overhead needed to support such a capability is kept at a reduced level.

The following is a mapping of independent claim 1 onto an example, non-limiting embodiment in the specification.[1]

1. A method of executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of a data processing apparatus, said method comprising the steps of:	Figures 1, 4, and 5, memory 54 and memory regions A and B.
(i) detecting an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region and said attempt triggering a memory abort;	Figure 3A, step 14, page 9, lines 7-23, page 12, lines 11-13, and current and following regions 44 and 46.
(ii) concatenating instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;	Figure 3A, steps 20 and 22, page 9, lines 25-32, page 12, lines 15-19, and fix-up memory 48 shown in Figure 4and shown also in 54 of Figure 5.
(iii) diverting program execution flow to execute said current variable length	Figures 3A and 3B, steps 24 and 30, page 10, lines 8-14, page 12,

[1] None of the claim mappings is intended to limit the claim scope, and none of the claim mappings is intended to be used in construing the meaning of claim terms.

instruction from within said concatenated instruction data in said fix-up memory address region; and	lines 19-25.
(iv) restoring program execution flow to execute instructions following said variable length instruction from within said following memory address region.	Figure 3B, steps 40 and 42, page 11, line 18-page 12, line 2, and page 13, lines 14-15.

The following is a mapping of independent claim 22 onto an example, non-limiting embodiment in the specification.

22. Apparatus for executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory, said apparatus comprising:	See Figures 1, 4, and 5, system elements 52 including memory 54 with memory regions A and B.
(i) a detector configured to detect an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region and said attempt triggering a memory abort;	Figure 3A, step 14, page 9, lines 7-23, page 12, lines 11-13, and current and following regions 44 and 46.
(ii) combining logic circuitry configured to concatenate instruction data from an end portion of said current memory	Figure 3A, steps 20 and 22, page 9, lines 25-32, page 12, lines 15-19, and fix-up memory 48 shown in

address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;	Figure 4and shown also in 54 of Figure 5.
(iii) diverting logic circuitry configured to divert program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and	Figures 3A and 3B, steps 24 and 30, page 10, lines 8-14, page 12, lines 19-25.
(iv) restoring logic circuitry configured to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.	Figure 3B, steps 40 and 42, page 11, line 18-page 12, line 2, and page 13, lines 14-15.

The following is a mapping of independent claim 43 onto an example, non-limiting embodiment in the specification.

43. A computer program product including a storage medium encoded with instruction code readable by a data processing apparatus, which when executed by the data processing apparatus, controls the data processing apparatus to execute a sequence of variable length instructions	See Figures 1, 4, and 5, system elements 52 including memory 54 with memory regions A and B.

stored within a plurality of discrete memory address regions within a memory of said data processing apparatus, said computer program product comprising:	
code configured after an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region and said attempt triggering a memory abort, said code including:	Figure 3A, step 14, page 9, lines 7-23, page 12, lines 11-13, and current and following regions 44 and 46.
(i) concatenating code encoded in the storage medium which, when executed by the data processing apparatus, controls the data processing apparatus to concatenate instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;	Figure 3A, steps 20 and 22, page 9, lines 25-32, page 12, lines 15-19, and fix-up memory 48 shown in Figure 4and shown also in 54 of Figure 5.
(ii) diverting code encoded in the storage medium which, when executed by the data processing apparatus, controls the data processing apparatus to divert program	Figures 3A and 3B, steps 24 and 30, page 10, lines 8-14, page 12, lines 19-25.

execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and	
(iii) restoring code encoded in the storage medium which, when executed by the data processing apparatus, controls the data processing apparatus to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.	Figure 3B, steps 40 and 42, page 11, line 18-page 12, line 2, and page 13, lines 14-15.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

All final rejections are requested for review on appeal. The questions to be decided by the Board include:

- whether the Examiner's rejection of claims 43-62 under 35 U.S.C. §101 is improper.
- whether the Examiner's rejection of claims 1-17, 21-38, 42-58, and 62 under 35 U.S.C. §103 based on Ohshima and Watt is improper.
- whether the Examiner's rejection of claims 18-20, 39-41, and 59-61 under 35 U.S.C. §103 based on Ohshima, Watt, and Komatsu is improper.

VII. ARGUMENT

A. The Rejection Under 35 U.S.C. §101: Claims 43-62 Recite Statutory Subject Matter[2]

Claim 43 recites: "A computer program product including a storage medium readable by a data processing apparatus encoded with instruction code which, when executed by the data processing apparatus, controls the data processing apparatus to execute a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of said data processing apparatus." The Examiner attempts to substitute the language from page 15, lines 6-13 in the specification for what is claimed—a clear error. A proper analysis under §101 is based on what is claimed. The quoted portion of claim 43 makes clear that the claim does not recite "pure software" because the product includes instruction code embodied in a storage medium executable by a data processing apparatus to achieve useful, concrete, and tangible results.

The Examiner's conclusion that "no useful, concrete and tangible results can be determined" is incorrect. The rejected claims are concerned with controlling a data processing apparatus to handle fragmented instructions (page 2, line 4-7 and page 3, lines 1-3). Although fragmented instructions can be desirable (e.g. for enhanced security), they also are associated problems such as memory aborts (see page 8, lines 21-24). The computer program product recited in claims

43-62 enables a data processing apparatus to handle such situations, and thus, to operate more efficiently. Certainly, given the central role that data processing apparatus have in daily activities of most people in the industrialized world, enabling data processing apparatus to operate more efficiently is a useful, concrete, and tangible result of claims 43-62.

The Examiner suggests that claims 43-62 are computer programs claimed as a computer listing per se. A review of the language used in claims 43-62 reveals that these claims are clearly not computer listings. Moreover, claim 43 includes language that describes a functional relationship with a storage medium readable by a data processing apparatus. The claim recites code elements, which when executed by the data processing apparatus, controls the data processing apparatus to perform useful, concrete, and tangible functions. These claimed functional interrelationships between the code and other elements of a computer (storage medium and data processor) permit the computer code's functionality to be realized.

The Examiner points to page 15, lines 6-13 in the specification to support the rejection. But a person of ordinary skill in the art would have understood that statements on page 15, lines 6-13 in the specification were made with the understanding that software-based control relates to a software program

[2] Numbered paragraph 5 on page 3 of the final action does not make sense in light of paragraph 2 on page 2 in the final action where the rejection of claims 22-42 under 35 U.S.C. §101 was withdrawn.

controlling a computer to perform an operation, and that hardware control relates to hardware performing an operation like fetching, decoding, and executing an instruction. See page 10, lines 21-24 of the specification: "At step 28, a single step flag is set within a register of a Javacard decoder controlling coprocessor CP14. Such control registers are generally conveniently accessible under software control (e.g. coprocessor register store instructions) and can be used by programs to pass configuration information to hardware." The registers, decoder, and coprocessor are hardware, and software control relates to instructions that control a processor.

The Examiner's position is out of touch with the reality of technology today in which computer program products form the foundation of so many modern day innovations. How else can an inventor protect against others selling commercial computer program products that utilize the inventor's innovations? Inventors have a right to stop such computer product infringers as well as infringers who actually use a computer controlled by that product. Indeed, searching for the term "computer program product" on the U.S. patent office's patent database returns over 20,000 granted U.S. patents that use that language. The claims of many of these patents use much the same format as that used in claim 43.

The Federal Circuit recently affirmed in *In Re Comiskey*, 499 F.3d 1365, 1379-80 (Fed. Cir. 2007) that a claim combining the use of a machine with a mental process recites patentable subject matter. The Comiskey court also relied

on its earlier *State Street* decision "holding patentable a 'system that allows an administrator to monitor and record the financial information flow and make all calculations necessary for maintaining a partner fund financial services configuration' where a 'computer or equivalent device [wa]s a virtual necessity to perform the task.'" *Id.* These cases contradict the Examiner's position and confirm that claims 43-62 recite statutory subject matter.

In an earlier office action dated July 24, 2007, the Examiner misapplied the Interim guidelines when referring to page 53. That section of those guidelines is directed *only* to a data structure representing descriptive material *per se* and computer programs representing computer listings per se. Claim 43 is neither. There is not even a single computer instruction included in claim 43. Moreover, later on page 53 of the Interim Guidelines, Examiners are instructed that "a claimed computer-readable medium encoded with a computer program is a computer element which defines structural and functional interrelationships between the computer program and the rest of the computer which permit the computer program's functionality to be realized, and is thus statutory." The language used in claim 43 very closely tracks the language quoted above and certainly recites a computer-readable medium encoded with a computer program. Thus, the very authority that the Examiner previously cited mandates that the computer-readable medium encoded with a computer program recited claim 43 is statutory.

The rejection of claims 43-62 under 35 U.S.C. §101 is improper and should be withdrawn.

B. The Rejection Of Claims 1-17, 21-38, 42-58, And 62 Under 35 U.S.C. §103 Based On Ohshima And Watt Is Improper

This application relates to executing a variable length instruction stored in distinct memory locations. The first part of a variable length instruction is stored in a first memory location, and the second part is stored in a second different memory location. A "fix-up" memory address region is used to bring together the two parts of the single instruction. When a two-part, variable-length instruction occurs, the program execution flow is temporarily diverted to the fix-up memory to read the freshly reconstructed variable length instruction stored there. Thereafter, the program execution flow is returned to reading from the normal memory.

Ohshima describes an instruction that includes both basic and expanded segments. A basic segment contains a code indicating the type of instruction (basic or expanded), and an expanded segment contains information relevant to the type of instruction specified by the basic segment. One instruction is formed from one or more basic and expanded segments. The length of expanded segments can vary, and this length is not known until its corresponding basic segment has been decoded. Hence, in a situation where one of Ohshima's instructions consists of two basic segments and the first basic segment is followed by an expanded

segment, the second basic segment cannot be input into a decoder until after the first basic segment has been decoded, which allows the length of the expanded segment to be determined. Thus, two cycles are required to decode the single instruction. See col. 2, line 59 to col. 3, line 9.

1. Ohshima And Watt Fail To Teach The Concatenation Into Fix-up Memory

Ohshima does not teach concatenating instruction data from two distinct memory address regions "into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction," as recited in claim 1. If there is a concatenation of instruction data in Ohshima, it is performed at the instruction code bus 14 which receives the rearranged segments from block 8 (see Figs. 6A and 6B). Applicants respectfully submit that a person skilled in the data processing art would not reasonably equate Ohshima's instruction code bus to the claimed fix-up memory region because a bus and a memory are two different things with two different functions and operations.

The Examiner refers to a "rearranged pointer P" in Figs. 4 and 8C. Such a pointer is not found in either of these figures. (Fig.1 shows a pointer P1 that points to the instruction code string C before it is rearranged on the instruction code bus.) Nor would a person skilled in the data processing art equate an address pointer with a fix-up memory region. A pointer can point to a memory address, but a pointer is not memory.

The Examiner further refers to Figs. 2a-2b. They should be understood in light of Fig. 1 which shows that the instruction code string segments are re-arranged onto the instruction code bus "by preliminarily separating the bus into the base segment fields ... and the expanded segment fields." Col. 5, lines 60-67 (emphasis added). In Fig. 2, the rearranged segments are provided from the instruction code bus to the instruction decoder 9 and instruction register 10. Fig. 2 "shows some examples of the arrangements of the expanded segments" where "the basic segment is connected to the upper side X of the bus and the expanded section outputs to the correspond expanded segment field Y [of the bus]." Col. 6, lines 10-14 (emphasis added). Any concatenating occurs on the instruction code bus and not in a fix-up memory.

2. Ohshima And Watt Fail To Teach Diverting Program Execution to Fix-up Memory And Then Restoring Program Execution

The Examiner maps the claimed diverting and restoring to execution of an instruction on Ohshima's instruction code bus and execution of the next instruction. But the instruction code bus is not part of the memory address space. Also, instruction decoding always takes place from Ohshima's instruction code bus (see Fig.1). So there is no need to divert program execution flow to the instruction code bus or restore program execution flow from the instruction code bus.

The Examiner refers to a rearranged pointer P in Figs. 4 and 8C, which again is not labeled in those Figures. The reader pointer 407 in Fig. 4 simply instructs which segments will be routed to specific instruction code bus lines 414. Read out pointer 7 in Figure 8B does the same thing via output position identification circuit 20, with the rearranged segments being output onto the instruction code bus 14_1 and 14_2. Program execution flow is not being diverted to another portion of memory to execute the rearranged instruction. As shown in Fig. 1, instruction execution always proceeds from the instruction code bus to the decoder 9 and instruction register 10.

Ohshima lacks the claimed restoring operation. Because there is no diversion to fix-up memory in Ohshima, there is no need for restoration of program execution flow from the fix-up memory. The Examiner refers to Fig. 5. But this figure further confirms that the rearranged instruction is provided to the instruction code bus before execution. See "arrangement on instruction code bus 520" at the bottom of Fig. 5.

3. The Rationale To Modify Ohshima Is Insufficient

The Examiner speculates that Watt's abort signal "could be implemented" in Ohshima to avoid "data loss due to improper memory address space access." But the question is why do that in Ohshima? There is no concern of improper memory access in Ohshima. And if such an abort was used in Ohshima, it would

likely come when the instruction code C is retrieved--not during the subsequent rearrangement of segments.

C. The Rejection Of Claims 18-20, 39-41, And 59-61 Under 35 USC §103 Based On Ohshima, Watt, And Komatsu Is Improper

Komatsu fails to remedy the deficiencies of Ohshima and Watt addressed above. Thus, the rejection of claims 18-20, 39-41, and 59-61 that depend on the main rejection based on Ohshima and Watt also cannot stand.

VIII. CONCLUSION

The Board should reverse the final rejections and order the application allowed.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____

John R. Lastova
Reg. No. 33,149

JRL/maa
Appendix A - Claims on Appeal

IX. CLAIMS APPENDIX

1. (Previously Presented) A method of executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of a data processing apparatus, said method comprising the steps of:

(i) detecting an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region and said attempt triggering a memory abort;

(ii) concatenating instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

(iii) diverting program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

(iv) restoring program execution flow to execute instructions following said variable length instruction from within said following memory address region.

2. (Original) A method as claimed in claim 1, wherein said steps of detecting is performed under hardware control.

3. (Original) A method as claimed in claim 1, wherein said steps of concatenating, diverting and restoring are performed under software control.

4. (Original) A method as claimed in claim 1, wherein variable length instructions are fetched from said memory to an instruction buffer before being executed.

5. (Original) A method as claimed in claim 4, wherein said step of detecting occurs as said variable length instruction is read from said instruction buffer.

6. (Original) A method as claimed in claim 4, wherein fetching of variable length instructions to said instruction buffer is performed by fetching instruction data from sequential memory addresses under hardware control.

7. (Original) A method as claimed in claim 6, wherein when a fetch is attempted from a memory address beyond an end point of said current memory address region to a buffer memory location, said buffer memory location is marked as not containing valid instruction data.

8. (Original) A method as claimed in claim 7, wherein said step of detecting comprises detecting an attempt to execute an instruction at least partially stored in a buffer memory location marked as not containing valid instruction data.

9. (Original) A method as claimed in claim 1, wherein a program counter value specifies a location of a variable length instruction to be executed.

10. (Original) A method as claimed in claim 9, wherein said steps of diverting and restoring act by modifying said program counter value.

11. (Original) A method as claimed in claim 1, comprising setting a single step flag, said single step flag serving to limit hardware execution of variable length instructions to a single variable length instruction.

12. (Original) A method as claimed in claim 11, wherein said single step flag serves to limit hardware execution of variable length instructions to a single variable length instruction from said concatenated instruction data to a single variable length instruction before control is returned to software to perform said step of restoring.

13. (Original) A method as claimed in claim 11, wherein upon hardware execution of said single variable length instruction, said single step flag is cleared under hardware control.

14. (Original) A method as claimed in claim 11, wherein said single step flag is stored within a coprocessor register.

15. (Original) A method as claimed in claim 1, comprising calculating a start address within said following region of memory of a following variable length instruction following said current variable length instruction.

16. (Original) A method as claimed in claim 15, wherein said step of calculating uses as inputs a start address of said following memory region and a program counter value pointing to said following variable length instruction within said fix-up memory region following execution of said current variable length instruction.

17. (Original) A method as claimed in claim 16, comprising storing said start address of said following memory region before said step of diverting.

18. (Original) A method as claimed in claim 1, wherein said variable length instructions are Javacard bytecode instructions executed as native instructions by said data processing apparatus.

19. (Original) A method as claimed in claim 18, wherein said data processing apparatus also supports execution of instructions of a further instruction set, said steps of concatenating, diverting and restoring being performed under control of instructions of said further instruction set.

20. (Original) A method as claimed in claim 19, wherein said steps of diverting and restoring are performed using state switching branch instructions that serve to switch to execution of Java bytecodes starting from a specified memory address location.

21. (Original) A method as claimed in claim 1, wherein a program to be executed is stored within fragmented memory regions within said memory.

22. (Previously Presented) Apparatus for executing a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory, said apparatus comprising:

(i) a detector configured to detect an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region and said attempt triggering a memory abort;

(ii) combining logic circuitry configured to concatenate instruction data from an end portion of said current memory address region and a start portion of said

following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

(iii) diverting logic circuitry configured to divert program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

(iv) restoring logic circuitry configured to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.

23. (Original) Apparatus as claimed in claim 22, wherein said detector is non-programmable hardware.

24. (Previously Presented) Apparatus as claimed in claim 22, wherein said concatenating logic circuitry, said diverting logic circuitry and said restoring logic circuitry comprise programmable hardware operating under software control.

25. (Original) Apparatus as claimed in claim 22, wherein variable length instructions are fetched from said memory to an instruction buffer before being executed.

26. (Original) Apparatus as claimed in claim 25, wherein said detector acts as said variable length instruction is read from said instruction buffer.

27. (Original) Apparatus as claimed in claims 25, wherein fetching of variable length instructions to said instruction buffer is performed by fetching instruction data from sequential memory addresses under hardware control.

28. (Original) Apparatus as claimed in claim 27, wherein when a fetch is attempted from a memory address beyond an end point of said current memory address region to a buffer memory location, said buffer memory location is marked as not containing valid instruction data.

29. (Previously Presented) Apparatus as claimed in claim 28, wherein said detector is configured to detect an attempt to execute an instruction at least partially stored in a buffer memory location marked as not containing valid instruction data.

30. (Original) Apparatus as claimed in claim 22, wherein a program counter value specifies a location of a variable length instruction to be executed.

31. (Previously Presented) Apparatus as claimed in claim 30, wherein said diverting logic circuitry and said restoring logic circuitry are configured to act by modifying said program counter value.

32. (Original) Apparatus as claimed in claim 22, comprising means for setting a single step flag, said single step flag serving to limit hardware execution of variable length instructions to a single variable length instruction.

33. (Original) Apparatus as claimed in claim 32, wherein said single step flag serves to limit hardware execution of variable length instructions to a single variable length instruction from said concatenated instruction data to a single variable length instruction before control is returned to software to perform said step of restoring.

34. (Original) Apparatus as claimed in claim 32, wherein upon hardware execution of said single variable length instruction, said single step flag is cleared under hardware control.

35. (Original) Apparatus as claimed in claim 32, wherein said single step flag is stored within a coprocessor register.

36. (Previously Presented) Apparatus as claimed in claim 22, comprising calculating logic configured to calculate a start address within said following region of memory of a following variable length instruction following said current variable length instruction.

37. (Previously Presented) Apparatus as claimed in claim 36, wherein said calculating logic circuitry is configured to use as inputs a start address of said following memory region and a program counter value pointing to said following variable length instruction within said fix-up memory region following execution of said current variable length instruction.

38. (Previously Presented) Apparatus as claimed in claim 37, comprising means for storing said start address of said following memory region before said diversion.

39. (Previously Presented) Apparatus as claimed in claim 22, wherein said variable length instructions are Java bytecode instructions executed as native instructions by said data processing apparatus.

40. (Previously Presented) Apparatus as claimed in claim 39, wherein said data processing apparatus also supports execution of instructions of a further instruction set, and wherein said concatenating logic circuitry, said diverting logic circuitry and said restoring logic circuitry are under control of instructions of said further instruction set.

41. (Previously Presented) Apparatus as claimed in claim 40, wherein said diverting logic circuitry and said restoring logic circuitry are configured to use mode switching branch instructions that serve to switch to execution of Java bytecodes starting from a specified memory address location.

42. (Original) Apparatus as claimed in claim 22, wherein a program to be executed is stored within fragmented memory regions within said memory.

43. (Previously Presented) A computer program product including a storage medium encoded with instruction code readable by a data processing apparatus, which when executed by the data processing apparatus, controls the data processing apparatus to execute a sequence of variable length instructions stored within a plurality of discrete memory address regions within a memory of said data processing apparatus, said computer program product comprising:

code configured after an attempt to execute a variable length instruction spanning two discrete memory address regions, said two discrete memory address regions being a current memory address region and a following memory address region and said attempt triggering a memory abort, said code including:

(i) concatenating code encoded in the storage medium which, when executed by the data processing apparatus, controls the data processing apparatus to concatenate instruction data from an end portion of said current memory address region and a start portion of said following memory address region into a fix-up memory address region of said memory to form concatenated instruction data containing said variable length instruction;

(ii) diverting code encoded in the storage medium which, when executed by the data processing apparatus, controls the data processing apparatus to divert program execution flow to execute said current variable length instruction from within said concatenated instruction data in said fix-up memory address region; and

(iii) restoring code encoded in the storage medium which, when executed by the data processing apparatus, controls the data processing apparatus to restore program execution flow to execute instructions following said variable length instruction from within said following memory address region.

44. (Previously Presented) A computer program product as claimed in claim 43, wherein detection of said attempt to execute a variable length instruction spanning two discrete memory address regions is performable under hardware control.

45. (Previously Presented) A computer program product as claimed in claim 43, wherein when the code is executed by the data processing apparatus, the data processing apparatus fetches variable length instructions from said memory to an instruction buffer before being executed.

46. (Previously Presented) A computer program product as claimed in claim 45, wherein when the code is executed by the data processing apparatus, said detection occurs as said variable length instruction is read from said instruction buffer.

47. (Previously Presented) A computer program product as claimed in claim 45, wherein when the code is executed by the data processing apparatus, the data processing apparatus fetches variable length instructions to said instruction buffer by fetching instruction data from sequential memory addresses under hardware control.

48. (Previously Presented) A computer program product as claimed in claim 47, wherein when the code is executed by the data processing apparatus and a fetch is attempted from a memory address beyond an end point of said current memory address region to a buffer memory location, the data processing apparatus marks said buffer memory location as not containing valid instruction data.

49. (Original) A computer program product as claimed in claim 48, wherein said detection comprises detecting an attempt to execute an instruction at least partially stored in a buffer memory location marked as not containing valid instruction data.

50. (Previously Presented) A computer program product as claimed in claim 43, wherein when the code is executed by the data processing apparatus, a program counter value specifies a location of a variable length instruction to be executed.

51. (Previously Presented) A computer program product as claimed in claim 50, wherein when the code is executed by the data processing apparatus, said diverting code and said restoring code act by modifying said program counter value.

52. (Previously Presented) A computer program product as claimed in claim 43, comprising setting code configured, when executed by the data processing apparatus, to set a single step flag, said single step flag serving to limit hardware execution of variable length instructions to a single variable length instruction.

53. (Original) A computer program product as claimed in claim 52, wherein said single step flag serves to limit hardware execution of variable length instructions to a single variable length instruction from said concatenated instruction data to a single variable length instruction before control is returned to software to perform said step of restoring.

54. (Previously Presented) A computer program product as claimed in claim 52, wherein upon hardware execution of said single variable length instruction, and the code is executed by the data processing apparatus, said single step flag is cleared under hardware control.

55. (Previously Presented) A computer program product as claimed in claim 52, wherein when the code is executed by the data processing apparatus, said single step flag is stored within a coprocessor register.

56. (Previously Presented) A computer program product as claimed in claim 43, comprising calculating code encoded in the storage medium which, when executed by the data processing apparatus, controls the data processing apparatus to calculate a start address within said following region of memory of a following variable length instruction following said current variable length instruction.

57. (Previously Presented) A computer program product as claimed in claim 56, wherein said calculating code which, when executed by the data processing apparatus, controls the data processing apparatus to use as inputs a start address of said following memory region and a program counter value pointing to said following variable length instruction within said fix-up memory region following execution of said current variable length instruction.

58. (Previously Presented) A computer program product as claimed in claim 57, wherein when the code is executed by the data processing apparatus, said start address of said following memory region is stored before said diversion.

59. (Previously Presented) A computer program product as claimed in claim 43, wherein said variable length instructions are Java bytecode instructions executed as native instructions by said data processing apparatus.

60. (Previously Presented) A computer program product as claimed in claim 59, wherein said data processing apparatus also supports execution of instructions of a further instruction set, and when the code is executed by the data processing apparatus, said concatenating, diverting and restoring are performed under control of instructions of said further instruction set.

61. (Previously Presented) A computer program product as claimed in claim 60, wherein said diverting code and said restoring code are configured to control the data processing apparatus to use state switching branch instructions that serve to switch to execution of Java bytecodes starting from a specified memory address location.

62. (Previously Presented) A computer program product as claimed in claim 43, wherein when the code is executed by the data processing apparatus, a program to be executed is stored within fragmented memory regions within said memory.

X. EVIDENCE APPENDIX

There is no evidence appendix.

XI. RELATED PROCEEDINGS APPENDIX

There is no related proceedings appendix.